Automata Learning meets Shielding

Martin Tappler[1,2], Stefan Pranger[1], Bettina Könighofer[1,3], Edi Muškardin[2,1], Roderick Bloem[1,2], and Kim Larsen[4]

[1] Graz University of Technology
[2] TU Graz-SAL DES Lab,Silicon Austria Labs, Graz, Austria
[3] Lamarr Security Research
[4] Aalborg University, Aalborg, Denmark
martin.tappler@ist.tugraz.at, stefan.pranger@iaik.tugraz.at, bettina.koenighofer@lamarr.at, edi.muskardin@silicon-austria.com, roderick.bloem@iaik.tugraz.at, kgl@cs.aau.dk

Abstract Safety is still one of the major research challenges in reinforcement learning (RL). In this paper, we address the problem of how to avoid safety violations of RL agents during exploration in probabilistic and partially unknown environments. Our approach combines automata learning for Markov Decision Processes (MDPs) and shield synthesis in an iterative approach. Initially, the MDP representing the environment is unknown. The agent starts exploring the environment and collects traces. From the collected traces, we passively learn MDPs that abstractly represent the safety-relevant aspects of the environment. Given a learned MDP and a safety specification, we construct a shield. For each state-action pair within a learned MDP, the shield computes exact probabilities on how likely it is that executing the action results in violating the specification from the current state within the next k steps. After the shield is constructed, the shield is used during runtime and blocks any actions that induce a too large risk from the agent. The shielded agent continues to explore the environment and collects new data on the environment. Iteratively, we use the collected data to learn new MDPs with higher accuracy, resulting in turn in shields able to prevent more safety violations. We implemented our approach and present a detailed case study of a Q-learning agent exploring slippery Gridworlds. In our experiments, we show that as the agent explores more and more of the environment during training, the improved learned models lead to shields that are able to prevent many safety violations.

Keywords: Automata Learning · Shielding · Markov Decision Processes

1 Introduction

Nowadays systems are increasingly autonomous and make extensive use of machine learning. The tremendous potential of autonomous, AI-based systems is contrasted by the growing concerns about their safety [11]. Their huge complexity makes it infeasible to formally prove their correctness or to cover the entire input space of a system with test cases. An especially challenging problem is ensuring safety during the learning process [21]. In model-free *reinforcement learning* (RL) [34], an agent aims to learn a task through trial-and-error via interactions with an unknown environment. While the model-free RL approach is very general as well as scalable and has successfully been applied in various challenging application domains [20], the learning agent needs to explore unsafe behavior in order to learn that it is unsafe.

Shielding [4] is a runtime enforcement technique that applies correct-by-construction methods to automatically compute shields from a given safety temporal logic specification [5] and a model that captures all safety-relevant dynamics of the environment. Shields have been categorized into post-shields and pre-shields. Post-shields monitor the actions selected by the agent and overwrite any unsafe action with a safe one. Pre-shields are implemented before the agent and block, at every time step, unsafe actions from the agent (also referred to as action masking). Thus, the agent can only choose from the set of safe actions. In this paper, we use pre-shielding since this setting allows the agent maximal freedom in exploring the environment.

In the *non-probabilistic setting* [6], shields guarantee that the safety specification will never be violated, working under the assumption that a complete and faithful environmental model of the safety-relevant dynamics is available. Shielding in the *probabilistic setting* [19], which is the standard setting in RL, assumes to have an environmental model in form of a Markov decision process (MDP) available. Given such an MDP $\mathcal{M}$ and a safety specification φ, the shield computes how likely it is that executing an action from the current state will result in violating φ within a given finite horizon. At any state s, an actions a is called *unsafe* if executing a incurs a probability of violating φ within the next k steps greater than a relative threshold λ w.r.t. the optimal safety probability possible in s. The resulting shield prohibits safety violations that can be prevented by planning ahead k steps into the future. Shielding requires a complete and accurate environmental model, but it is rarely the case that such a model is available. However some data about the environment often exists, for example, a RL agent collects data by exploring the environment.

Automata learning [38,1,17] is a well-established technique to automatically learn automata models of black-box systems from observed data. The data used for automata learning is usually given in the form of observation traces, which are sequences of observations of the environment's state and actions chosen by the agent. *Passive MDP learning* [23,24] is able to learn MDP models from a multiset of sampled observation traces. Thus, the learned MDP depends on the given sampled traces.

Our approach. In this paper, we consider the setting of RL in an initially unknown environment. The goal is to reduce safety violations during the exploration phase of the RL-agent by combining *passive MDP learning* with *probabilistic shielding* in an iterative approach. Initially, the MDP representing the environment is unknown. During runtime, the agent collects observation traces while exploring the environment. After having a large enough initial multiset of traces, we first transform the sequences of observed states in the traces into observations, which include only the safety-relevant information, using a suitable abstraction function. The abstract traces are then used to learn a first estimate of the safety-relevant MDP. From this initial MDP and a given safety specification, we construct an initial shield. After the shield is constructed, the agent is augmented with the shield, i.e., the shield blocks unsafe actions from the agent and the agent can pick from the set of safe actions. The newly collected traces are added to the multiset of all traces. After collecting a predefined number of new traces, our approach learns a new safety-relevant MDP from the multiset of all collected traces and creates a new shield.

At every iteration, the shield is built from an MDP that approaches more and more the real MDP underlying the environment modulo the abstraction to safety-relevant observations. Thus, the resulting shields are getting more informed and prevent the agent from entering more safety-critical situations.

Outline. The rest of the paper is structured as follows. We present the related work in Section 2 and discuss the relevant foundations in Section 3. We present our approach for safe learning via shielding and automata learning in Section 4. We present our experimental results in Section 5 and conclude in Section 6.

2 Related Work

We combine automata learning and probabilistic verification to create safety shields in our approach. Early work on such combinations has been performed by Cobleigh et al. [10], who propose to learn assumptions for compositional reasoning. More closely related to our work is black box checking by Peled et al. [31], where they present a technique for model checking of deterministic black-box systems. Learning-based testing by Meinke and Sindhu [26] follows a similar approach of incremental learning of hypothesis models and model checking of these hypotheses. In previous work [3], we proposed a technique inspired by black box checking for probabilistic reachability checking of stochastic black-box systems. As in this paper, we applied IOAlergia [23,24] to learn MDPs. Rather than computing safety shields from learned MDPs, we computed policies to satisfy reachability objectives. We also proposed a technique for L^*-based learning of MDP [35], which may serve as a basis for RL and shielding. In this paper, we combine stochastic learning and abstraction with respect to safety-relevant features to improve RL. Nouri et al. [30] also apply abstraction on traces with respect to properties of the system under consideration in order to learn abstract probabilistic models. In contrast to us, they aim to improve the runtime of statistical model-checking and they learn Markov chains that are not controllable via inputs.

Recently, various authors have proposed combinations of automata learning and reinforcement learning [14,42,18,13]. By learning finite-state models, such as so-called reward machines, they provide additional high-level structure for RL. This enables RL when rewards are non-Markovian, i.e., the gain depends not only on the current state and action, but on the path taken by the agent. DeepSynth [16] follows a similar approach to improve RL with sparse rewards.

Related to these approaches, Muskardin et al. [29] propose a combination of reinforcement learning and automata learning to handle partial observability, i.e., non-Markovian environments.

Fu and Topcu [12] presented an approach for a learning-based synthesis of policies for MDPs w.r.t. temporal logic specifications that are probably approximately correct. In contrast to us, they assume the topology of the MDP to be known, so that only transition probabilities need to be learned.

Alshiekh et al. [4] proposed shielding for RL. Jansen et al. [19] proposed the first method to compute safety shields using a bounded horizon in MDPs. Giacobbe et al. [15] applied the same technique on 31 Atari 2600 games. The approach was further extended by Könighofer et al. [22]. Instead of analyzing the safety of all state-action pairs ahead of time, the approach uses the time between two successive decisions of an agent to analyze the safety of actions on the fly. Pranger et al. [33] proposed an iterative approach to shielding that updates the transition probabilities of the MDP based on observed behavior and computes new shields in regular intervals. To construct our shields, we use the approach proposed by Jansen et al. [19]. Similarly to Pranger et al. [33], we iteratively construct new shields, but do not rely on a known topology of the MDP.

As in our approach, Waga et al. [40] use automata learning to dynamically construct shields during runtime. The main difference to our work is that they assume that the environment behaves deterministically, whereas we allow probabilistic environmental behavior which is the standard assumption in reinforcement learning.

3 Preliminaries

Basics. Given a set E, we denote by $Dist(E)$ the set of probability distributions over E, i.e. for all μ in $Dist(E)$ we have $\mu : E \to [0,1]$ such that $\sum_{e\in E} \mu(e) = 1$. In Section 4, we apply two randomized functions *coinFlip* and *randSel*. The function *coinFlip* is defined by $\mathbb{P}(coinFlip(p) = \top) = p$ and $\mathbb{P}(coinFlip(p) = \bot) = 1-p$ for $p \in [0,1]$. The function *randSel* samples an element e from a given set E according to uniform distribution, i.e., $\forall e \in E : \mathbb{P}(randSel(E) = e) = \frac{1}{|E|}$.

3.1 Markov Decision Processes and Reinforcement Learning

Definition 1. *A* ***Markov decision process (MDP)*** *is a tuple* $\langle \mathcal{S}, s_0, \mathcal{A}, \mathcal{P}\rangle$ *where* $\mathcal{S}$ *is a finite set of states,* $s_0 \in \mathcal{S}$ *is the initial state,* $\mathcal{A}$ *is a finite set of actions, and* $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to Dist(\mathcal{S})$ *is the probabilistic transition function.*

For all $s \in \mathcal{S}$ the available actions are $\mathcal{A}(s) = \{a \in \mathcal{A} \mid \exists s', \mathcal{P}(s,a)(s') \neq 0\}$ and we assume $|\mathcal{A}(s)| \geq 1$. We associate an MDP $\mathcal{M}$ with a *reward function* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.

Traces. A finite *path* ρ through an MDP is an alternating sequence of states and actions, i.e. $\rho = s_0 a_1 s_1 \cdots a_{n-1} s_{n-1} a_n s_n \in s_0 \times (\mathcal{A} \times \mathcal{S})^*$. The set of all paths of an MDP $\mathcal{M}$ is denoted by $Path_{\mathcal{M}}$. We refer to a path augmented with the gained reward as a *reward trace* $\tau = s_0 a_1 r_1 s_1 \cdots s_{n-1} a_n r_n s_n$ with $r_i = \mathcal{R}(s_{i-1}, a_i, s_i)$. In the remainder of the paper, we treat reward traces also as sequences of triples (a_i, r_i, s_i) comprising an action, the gained reward, and the reached state.

Policies. A memoryless policy defines for every state in an MDP a probability distribution over actions. Given an MDP $\mathcal{M} = \langle S, A, s_0, \mathcal{P}\rangle$, a *memoryless policy* for $\mathcal{M}$ is a function $\sigma : \mathcal{S} \to Dist(\mathcal{A})$. A *memoryless deterministic policy* $\sigma : \mathcal{S} \to \mathcal{A}$ is a function over action given states.

Reinforcement Learning. An RL agent learns a task through trial-and-error via interactions with an unknown environment. The agent takes *actions* and receives feedback in form from *observations* on the state of the environment and *rewards*. The goal of the agent is to maximize the expected accumulated reward.

Typically, the environment is modeled as an MDP $\mathcal{M} = \langle \mathcal{S}, s_0, \mathcal{A}, \mathcal{P}\rangle$ with associated reward function $\mathcal{R}$. At each step t of a training episode, the agent receives an observation s_t. It then chooses an action $a_{t+1} \in \mathcal{A}$. The environment then moves to a state s_{t+1} with probability $\mathcal{P}(s_t, a_{t+1})(s_{t+1})$. The reward is determined with $r_{t+1} = \mathcal{R}(s_t, a_{t+1}, s_{t+1})$. We refer to negative rewards $r_t < 0$ as *punishments*. The *return* $\texttt{ret} = \Sigma_{t=1}^{\infty} \gamma^t r_t$ is the cumulative future discounted reward, where r_t is the immediate reward at time step t, and $\gamma \in [0, 1]$ is the discount factor that controls the influence of future rewards. The objective of the agent is to learn an *optimal policy* $\sigma^\star : \mathcal{S} \to \mathcal{A}$ that maximizes the expectation of the return, i.e. $\max_{\sigma \in \Sigma} \mathbb{E}_\sigma(\texttt{ret})$. A training episode ends after a maximum episode length of t_{max} steps.

Q-learning is one of the most established RL algorithms. The Q-function for policy σ is defined as the expected discounted future reward gained by taking an action a from a state s and following policy σ thereafter. Tabular Q-learning [41] uses the experience (s_t, a_t, r_t, s_{t+1}) to learn the Q-function $Q^\star(s, a)$ corresponding to an optimal policy $\sigma^\star(s, a)$. The update rule is defined as

$$Q(s_i, a_{i+1}) \leftarrow (1 - \alpha) \cdot Q(s_i, a_{i+1}) + \alpha(r_{i+1} + \gamma \cdot \max_{a \in A}(Q(s_{i+1}, a))),$$

where α is the learning rate and γ is the discount factor.

Definition 2. *A* ***deterministic labeled MDP*** $\mathcal{M}_L = \langle \mathcal{S}, s_0, \mathcal{A}, \mathcal{P}, L\rangle$ *is an MDP with a labeling function* $L : \mathcal{S} \to O$ *mapping states to observations from a finite set* O*. The transition function* $\mathcal{P}$ *must satisfy the following determinism property:* $\forall s \in \mathcal{S}, \forall a \in \mathcal{A} : \delta(s, a)(s') > 0 \wedge \delta(s, a)(s'') > 0$ *implies* $s' = s''$ *or* $L(s') \neq L(s'')$.

In this paper, we use passive automata learning to compute abstract MDPs of the environment in the form of deterministic labeled MDPs. These MDPs represent safety-related information only and will not be used for RL but for shielding. Therefore, there is no need to use rewards in combination with deterministic labeled MDPs.

Given a path ρ in a deterministic labeled MDP $\mathcal{M}_L$. Applying the labeling function on all states of a path ρ results in a so called *observation trace* $L(\rho) = L(s_0) a_1 L(s_1) \cdots a_{n-1} L(s_{n-1}) a_n L(s_n)$. Note that due to determinism, an observation trace $L(\rho)$ uniquely identifies the corresponding path ρ.

3.2 Learning of MDPs

We learn deterministic labelled Markov decision processes (MDPs) via IOAlergia [23,24], which is an adaptation of Alergia [8]. IOAlergia takes a multiset $\mathcal{T}_o$ of observation traces as input and first constructs a tree representing the observation traces, by merging common prefixes. The tree has edges labeled with actions and nodes that are labeled with observations. Each edge corresponds to a trace prefix with the label sequence that is visited by traversing the tree from the root to the edge. Additionally, edges are associated with frequencies that denote how many traces in $\mathcal{T}_o$ have the trace corresponding to an edge as a prefix. Normalizing these frequencies would already yield tree-shaped MDP.

For generalization, the tree is transformed into an MDP with cycles through an iterated merging of nodes. Two nodes are merged if they are compatible, i.e., their future behavior is sufficiently similar. For this purpose, we check whether the observations in the sub-trees originating in the nodes are not statistically different. The parameter $\epsilon_{\text{Alergia}}$ controls the significance level of the applied statistical tests. If a node is not compatible with any other potential node, it is promoted to an MDP state. Once all potential pairs of nodes have been checked, the final deterministic labeled MDP is created by normalizing the frequencies on the edges to yield probability distributions for the transition function $\mathcal{P}$. In this paper, we refer to this construction as MDP learning and we denote calls to IOAlergia by $\mathcal{M}_{\bigcup} = \text{IOAlergia}(\mathcal{T}_o, \epsilon_{\text{Alergia}})$, where $\mathcal{M}_{\bigcup}$ is the deterministic labeled MDP learned from the multiset of observation traces $\mathcal{T}_o$.

3.3 Shielding in MDPs

Specifications and Model Checking. We consider specifications given in the safety fragment of linear temporal logic (LTL) [5]. For an MDP $\mathcal{M}$ and a safety specification φ, probabilistic model checking employs linear programming or value iteration to compute the probabilities of all states and actions of the $\mathcal{M}$ to satisfy an φ. Specifically, the probabilities $\eta^{\max}_{\varphi,\mathcal{M}}: \mathcal{S}\times\mathbb{N} \to [0,1]$ or $\eta^{\min}_{\varphi,\mathcal{M}}: \mathcal{S}\times\mathbb{N} \to [0,1]$ give for all states the maximal (or minimal) probability over all possible policies to satisfy φ, within a given number of steps. For instance, a safety property $\varphi = \mathbf{G}(\neg\mathcal{S}_{unsafe})$ could encode that a set of unsafe states $S_{unsafe} \in \mathcal{S}$ must not be entered. Then $\eta^{\max}_{\varphi,\mathcal{M}}(s,h)$ is the maximal probability to not visit S_{unsafe} from state $s \in \mathcal{S}$ in the next h steps.
Shield Construction. Given an MDP $\mathcal{M}$, a safety specification φ, and a finite horizon h, the task of the shield is to limit the probability to violate the safety specification φ within the next h steps.

For any state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$, the *safety-value* $val_{\varphi,\mathcal{M}}(s,a,h)$ is computed which gives the maximal probability to stay safe from s after executing a, i.e.,

$$val_{\varphi,\mathcal{M}}(s,a,h) = \eta^{\max}_{\varphi,\mathcal{M}}(\mathcal{P}(s,a), h-1).$$

The *optimal safety-value* $optval_{\varphi,\mathcal{M}}(s)$ of s is the maximal safety value of any action a in state s within the next $h-1$ steps, i.e.,

$$optval_{\varphi,\mathcal{M}}(s,h) = \max_{a\in\mathcal{A}(s)} val_{\varphi,\mathcal{M}}(s,a,h) = \eta^{\max}_{\varphi,\mathcal{M}}(\mathcal{P}(s,a), h-1).$$

An action a in s is *unsafe* if the safety value of a is lower than the optimal safety-value by some threshold λ_{sh}, i.e., an action a in state s is *unsafe* iff

$$val_{\varphi,\mathcal{M}}(s,a,h) < \lambda_{sh} \cdot optval_{\varphi,\mathcal{M}}(s,h).$$

We refer to actions that are not unsafe as safe actions.

The task of the shield is to block any unsafe action from the agent, thereby restricting the set of available actions $\mathcal{A}(s)$ to the set of safe actions. A shield is a relation $\pi_{\bigtriangledown} : \mathcal{S} \to 2^{\mathcal{A}(s)}$ allowing at least one action for any state.

4 Learned Shields for Safe RL

In this section, we present our iterative approach for safe reinforcement learning via automata learning and shielding. We first discuss the setting in which the RL agents operates and give our problem statement. Then we give an overview of our approach. Finally, we discuss the individual steps of our approach in detail.

4.1 Setting and Problem Statement

Setting. We consider an RL agent acting in an unknown environment that can be modeled as an MDP $\mathcal{M} = \langle \mathcal{S}, s_0, \mathcal{A}, \mathcal{P} \rangle$ with an associated reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. However, since the environment is unknown at the beginning of the learning phase, the agent has no knowledge about the structure of $\mathcal{M}$.

We assume that the safety critical properties are given in form of an LTL formula φ. Without knowledge about the safety-relevant dynamics of the environment, it is not possible to prohibit violating φ while the RL agent is exploring the environment.

During the exploration phase of the RL agent, a multiset of reward traces is collected. We assume to have an observation function $Z : \mathcal{S} \to O$ given that maps any state $s \in \mathcal{S}$ states to a safety-relevant observation $o \in O$.
Problem Statement. Consider the setting as discussed above. The goal of our approach is to use the available data from the environment in form of collected traces and the given safety specification to prevent safety violations if possible.

4.2 Overview of Iterative Safe RL via Learned Shields

Our approach combines automata learning, shielding, and reinforcement learning in an iterative manner. Our approach performs n_{iter} iterations. At the first iteration with $i = 1$, we start from an empty multiset of reward traces $\mathcal{T} = \emptyset$, an initial learned $\mathcal{M}_{\bigtriangledown 0}$ with a single state and a self-loop for any action, and an initial shield $\pi_{\bigtriangledown 0}$ that allows at every step any action.

At each iteration i of n iterations, our approach works as follows:

Step 1 - Exploration. The RL agent explores the environment to learn the optimal policy. At each iteration i, the agent trains for a number of $n_{episodes}$. The agent is augmented by a shield $\pi_{\bigtriangledown i-1}$ that restricts its available actions.

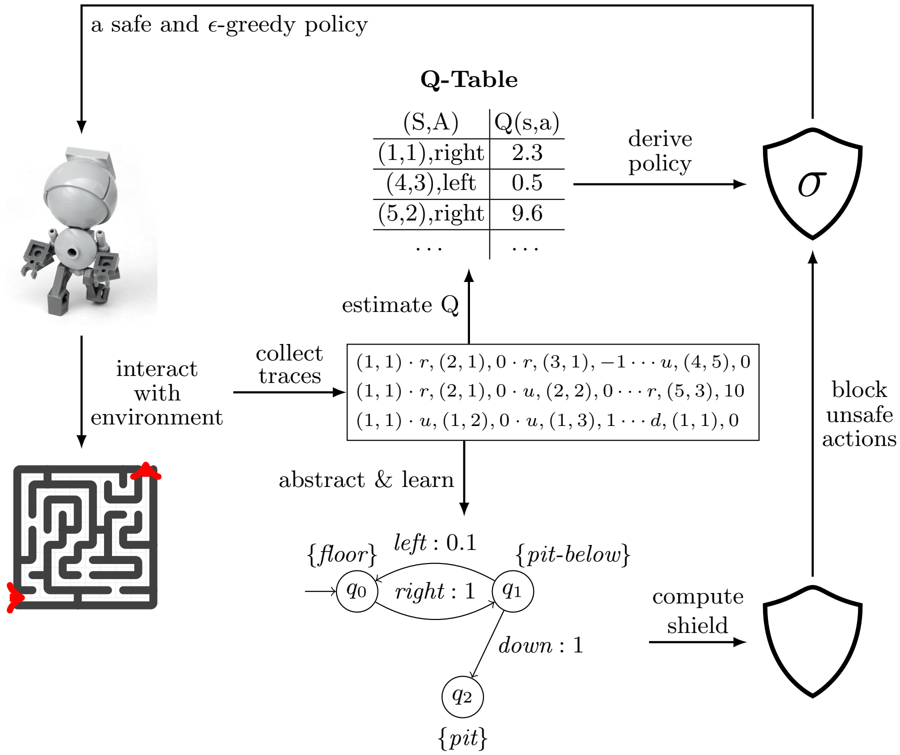

Figure 1. Iterative safe reinforcement learning via learned shields.

The learned MDP $\mathcal{M}_{\triangledown i-1}$ is simulated in parallel during the exploration. Each training episode yields a reward trace τ that is added to the multiset of all collected traces $\mathcal{T}$, i.e., $\mathcal{T}' = \{\tau\} \uplus \mathcal{T}$.

Step 2 - MDP Learning. After executing $n_{episodes}$ episodes, we learn a deterministic labeled MDP $\mathcal{M}_{\triangledown i}$ from $\mathcal{T}$.

Step 3 - Shield Construction. Using $\mathcal{M}_{\triangledown i}$, a given specification φ, and a finite horizon h, we compute a shield $\pi_{\triangledown i}$ and continue in **Step 1**.

Figure 1 provides a graphical overview of the proposed approach. Based on this figure, we discuss the individual steps of our approach in detail. For each iteration $1 \leq i \leq n_{iter}$ our approach performs the following steps.

Step 1 - Exploration. The agent interacts with an unknown stochastic environment $\mathcal{M} = \langle S, s_0, \mathcal{A}, \mathcal{P} \rangle$, depicted by a maze. The agent picks actions that are considered to be safe by the current shield and receives observations of the state of the environment and rewards. At iteration i, we have given the set of collected reward traces $\mathcal{T}$, the learned MDP $\mathcal{M}_{\triangledown i-1} = \langle S_\triangledown, s_{\triangledown 0}, \mathcal{A}_\triangledown, \mathcal{P}_\triangledown, L \rangle$, and the current shield $\pi_{\triangledown i-1}$. Each episode resets the environment and starts in a fixed initial state $s_0 \in \mathcal{S}$ for $\mathcal{M}$ and $s_{\triangledown 0} \in \mathcal{S}_\triangledown$ for $\mathcal{M}_\triangledown$. At every step t, the agent observes $s_t \in \mathcal{S}$. Based on the observed label $Z(s_t)$, the learned MDP $\mathcal{M}_{\triangledown i-1}$ moves to state $s_{\triangledown t}$. We give in Section 4.3 the details on how to simulate $\mathcal{M}_{\triangledown i}$.

The shield determines the set of safe actions $\mathcal{A}_{\triangledown} = \pi_{\triangledown i-1}(s_{\triangledown t})$ and sends it to the agent. The agent selects a next action $a_{t+1} \in \mathcal{A}_{\triangledown}(s)$. The environment executes a_{t+1} and sends the agent a reward r_{t+1} and state observation s_{t+1}. Based on $(s_t, a_{t+1}, r_{t+1}, s_{t+1})$, the agent performs a policy update. Figure 1 represents the learned Q-function of the RL agent as a Q-table, from which we derive an ϵ-greedy policy for training. Please note that our approach is general and applicable to deep Q-learning as well as to tabular Q-learning.

A training episode ends after a maximal number of t_{max} steps. It may end earlier in case of violating safety or performing the task that needs to be learned (e.g., reaching a certain goal state). Each episode yields a reward trace $\tau = s_0 a_1 r_1 s_1 \cdots s_{n-1} a_n r_n s_n$ that is added to the multiset $\mathcal{T}$ of all collected traces, i.e., $\mathcal{T}' = \{\tau\} \uplus \mathcal{T}$.

Step 2 - MDP Learning. After $n_{episodes}$ training episodes, resulting in a new reward trace per episode, we learn a new model of the environment dynamics. Given the multiset of reward traces $\mathcal{T}$ observed while exploring the environment, we abstract away all information in the traces that is not relevant to safety. For each reward trace $\tau = s_0 a_1 r_1 s_1 \cdots s_{n-1} a_n r_n s_n \in \mathcal{T}$, we first discard the rewards to obtain a path $\rho = s_0 a_1 s_1 \cdots s_{n-1} a_n s_n$ and second apply the abstraction function Z to obtain an observation trace $\tau_o = Z(\rho) = Z(s_0) a_1 Z(s_1) \cdots Z(s_{n-1}) a_n Z(s_n)$. For example, the states in the path ρ may represent the exact coordinates of the agent's positions and distance measurements. Relevant for safety might only be the distances. In such a case, the abstraction function Z would abstract away the concrete position and only keep the distances.

To compute the deterministic labeled MDP $\mathcal{M}_{\triangledown i}$,we invoke IOAlergia by calling IOAlergia$(\mathcal{T}_0, \epsilon_{\text{Alergia}})$, where $\epsilon_{\text{Alergia}}$ is a parameter specifying the significance level of statistical tests performed by IOAlergia.

Additionally, we make $\mathcal{M}_{\triangledown i}$ *action-complete*. During the training phase, we propose to make $\mathcal{M}_{\triangledown i}$ with $i < n$ action-complete by adding self-loop transitions to all state-action pairs $(s_{\triangledown}, a_{\triangledown})$ where $\mathcal{P}_{\triangledown}(s_{\triangledown}, a_{\triangledown})$ is not defined. That is, we set for all such state-action pairs $\mathcal{P}_{\triangledown}(s_{\triangledown}, a_{\triangledown}) = \{s_{\triangledown} \mapsto 1\}$. The rationale behind this is that whenever an action's effect is unknown, we assume that it leaves the safety of the corresponding state unchanged.

For the final shield $\pi_{\triangledown}$ that will be used permanently after training, we propose a more conservative approach and to make $\mathcal{M}_{\triangledown i}$ with $i = n$ action-complete by adding a special sink state $s_{\neg\varphi}$ that violates φ and adding transitions to $s_{\neg\varphi}$. In the training phase, the resulting shield $\pi_{\triangledown}$would not be suitable since it would prohibit exploration. For the exploration phase, however, the behavior of $\pi_{\triangledown}$may be desirable since it blocks behavior that has not been explored sufficiently.

Step 3 - Shield Construction. The abstract MDP $\mathcal{M}_{\triangledown i}$ encodes the safety-relevant information about the environment. We use $\mathcal{M}_{\triangledown i}$, the safety specification φ, and a finite horizon h, to compute the safety values of all state-action pairs in $\mathcal{M}_{\triangledown i}$. Based on a given relative threshold λ_{sh}, we compute a shield $\pi_{\triangledown i}$

Algorithm 1 A single RL training's episode using a learned shield.

Input: Q-function Q, a learned deterministic labeled MDP $\mathcal{M}_{\shield} = \langle S_{\shield}, \mathcal{A}_{\shield}, s_{\shield 0}, \mathcal{P}_{\shield}, L \rangle$, exploration rate ϵ, safety shield $\pi_{\shield}$, environment with **step** and **reset**

Output: Updated Q-function Q

1: $s_{\shield} \leftarrow s_{\shield 0}$
2: $s \leftarrow$ **reset**
3: $t \leftarrow 0$ ▷ steps
4: **while** $t < t_{max}$ **do**
5: $\quad \mathcal{A}_{\shield} \leftarrow \pi_{\shield}(s_{\shield})$ ▷ safe actions for s
6: $\quad$ **if** $coinFlip(\epsilon)$ **then**
7: $\qquad a \leftarrow randSel(\mathcal{A}_{\shield})$
8: $\quad$ **else**
9: $\qquad a \leftarrow \operatorname{argmax}_{a' \in \mathcal{A}_{\shield}} Q(s, a')$
10: $\quad r, s' \leftarrow$ **step**(a) ▷ step in environment
11: $\quad s_{\shield} \leftarrow s'_{\shield}$ where $\mathcal{P}(s_{\shield}, a)(s'_{\shield}) > 0$ and $L(s'_{\shield}) = Z(s')$ ▷ step in learned MDP
12: $\quad$ UPDATEQ(s, a, r, s')
13: $\quad$ **if** s' is a terminal state **then**
14: $\qquad$ **break**
15: $\quad t \leftarrow t + 1$

which allows for any state all actions that are safe w.r.t. λ_{sh} and the optimal safety value. After constructing the shield $\pi_{\shield i}$, the current shield of the agent is set to $\pi_{\shield i}$. The agent continues to explore the environment and learn the optimal strategy using $\pi_{\shield i}$ (Step 1).

4.3 Details for Training using Learned Shields

In this section we discuss a single RL episode of an agent augmented with a learned shield in detail. The pseudocode is given in Algorithm 1. The RL agent interacts with the environment through two operations: **reset** and **step**[5].

reset: This operation resets the environment to a fixed initial state. This state s_0 from the unknown environment MDP is also returned from **reset**.

step: The operation **step**(a) takes an action a and executes a causing a probabilistic state transition. It returns a pair r, s', where r is the reward gained by performing a and s' is the reached state when executing a.

The algorithm starts with initializing the learned MDP state as well as the environment state (Line 1 and 2). Then we enter a loop in which the agent performs a maximum of t_{max} steps.

The RL agent applies ϵ-greedy learning, i.e., it explores a random action with probability ϵ and otherwise performs the optimal action according to the RL agent's current knowledge. Note that both, random exploration and exploitation, are shielded, i.e., actions are chosen from the set of safe actions.

[5] the convention of OpenAI gym [7]

For every step, the shield provides a set of safe actions (Line 5). With probability ϵ, the RL agent selects a random safe action in Line 7. Otherwise, it determines the currently optimal action in Line 9. The chosen action is executed in Line 10. In Line 11, the state of $\mathcal{M}_{\bigtriangledown}$ is updated. To conclude the training step, we update the agent's Q-table.

If the agents visits a terminal state, the loop terminates before performing t_{max} steps (Line 13). A terminal state may, for instance, be reached by completing the task to be solved or by violating safety.

Execution phase. After training, we use the same approach to execute an agent, but with ϵ set to 0, such that only safe and optimal actions are executed.

5 Experiments

In our experimental evaluation, we evaluated our approach on 24 slippery gridworld environments of varying shapes and sizes. We implemented a tabular Q-learning agent that should learn to reach a given goal state quickly while staying safe. We learn deterministic labeled MDPs using IOAlergia implemented in AALpy [28]. The shields are created from the learned MDPs using the shield synthesis tool Tempest [32]. In our experiments, we discuss the scalability of our approach and its effectiveness by comparing the averaged gained reward during training with and without learned shields.

Experimental setup. All experiments have been executed on a desktop computer with a 4 x 2.70 GHz Intel Core i7-7500U CPU, 7.7 GB of RAM running Arch Linux.

Availability. An implementation of our framework for iterative safe RL via learned shields is available online, together with several examples and detailed execution instructions to reproduce our results[6]. The implementation includes a shielded tabular Q-learning agent written in Python.

Usability. Our prototype implementation allows users to easily perform their own experiments. AALpy's IOAlergia implementation has a simple text-based interface and Tempest uses the well-established PRISM MDP format that is also supported by AALpy, as well as a property language similar to PRISM and Storm. Furthermore, our prototype implementation can be easily extended. The tools, AALpy and Tempest, can be used as black boxes. Therefore, there is no need to know the implementation details of these tools. The only caveat is that Tempest is easiest to use through a Docker container.

5.1 Case Study Subjects

Gridworlds. We used three types of parameterized gridworlds, the smallest instances are depicted in Figure 2, Figure 3 and Figure 4. Each gridworld has a dedicated *start* tile and a *goal* tile, marked by *Entry* and *Goal* on the left-hand side of the figures. A gridworld might also have *intermediate-goal* tiles. If a tile

[6] `https://github.com/DES-Lab/Automata-Learning-meets-Shielding`

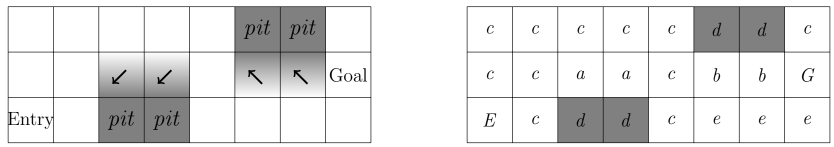

Figure 2. The smallest of the zigzag gridworlds.

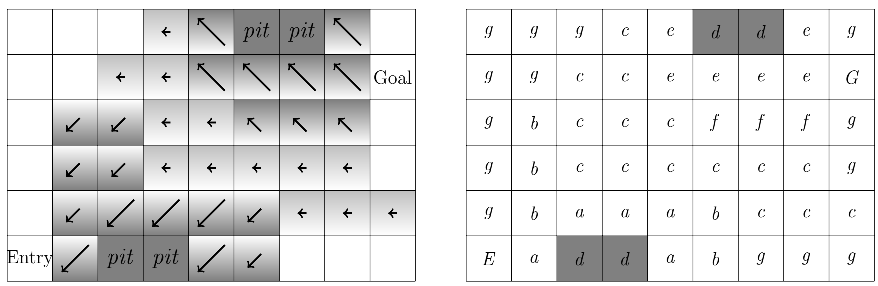

Figure 3. The smallest of the slippery shortcut gridworlds.

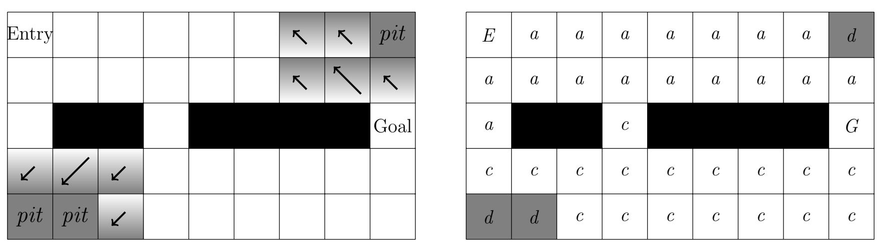

Figure 4. The smallest of the wall gridworlds.

is marked as *pit*, then this tile marks an unsafe location and the agent is not allowed to visit this tile. Black tiles represent *walls* that restrict movement but are not safety critical. Additionally, each tile has a *terrain*, denoted by lowercase letters on the right-hand side of the figures. If a tile is "slippery", the tile is labeled with an arrow and grey gradients on the left-hand side of the figures. If an agent tries to move from a slippery tile, the intended movement of the agent might be altered into the direction of the arrow with a specific tile-dependent probability. The length of arrows corresponds to the probability of slipping. That is, a long arrow pointing downward left means that the agent is very likely to slip either to the left or downwards. Additionally, every tile has (x, y) coordinates.

Gridworld Shapes. Next, we discuss each of the three gridworld shapes briefly, where Figures 2 to 4 show the smallest gridworld of each shape. To get insights into how the state space affects learning, we vary each type of gridworld in size by creating eight versions of increasing size.

- *zigzag*: In the *zigzag* gridworlds illustrated in Figure 2, we have pairs of pit tiles located in alternation at the bottom and the top of the map with a fixed distance between adjacent pit pairs. The goal is placed so that the agent could move along a straight line from left to right, but it would travel across slippery tiles next to the pits. The shield thus helps to avoid these dangerous tiles to perform zigzag walks to the goal.
- *slippery shortcuts*: The *slippery shortcuts* gridworlds illustrated in Figure 3 are similar to the *zigzag* maps in that the agent could take shortcuts across slippery tiles next to pits. In this case, the probability of unsuccessful moves decreases with increasing distance from the pits. Hence, an optimal policy has to find a balance between taking risks and gaining higher reward due to shorter paths.
- *walls*: In the *walls* gridworlds illustrated in Figure 4, the agents must find a way from the start to the goal by navigating around walls and pits that block the shortest paths. There are fewer slippery tiles.

Reinforcement Learning. We implemented a tabular Q-learning agent. The agent's task is to navigate from start to goal by moving into one of the four cardinal directions at each time step.

Every training episode, the agent collects reward traces of the form $\tau = (x_0, y_0), a_1, r_1, (x_1, y_1) \dots a_n, r_n, (x_n, y_n)$ with x_i and y_i representing the x and y coordinates of the tiles. The set of available *actions* comprises of the four actions {left, right, up, down} to move into the corresponding direction. The *reward function* of the agent is defined as follows:

- If the agent reaches the goal within less than t_{max} steps, it receives a reward of +100. Additionally, reaching the goal ends the training episode.
- If the agent reaches a tile with a *pit*, it receives a punishment of −100 and the training episode ends.
- Additionally, the agent receives a reward of −0.5 per step.
- Gridworlds might have intermediate goal states that are rewarded with +20 but do not terminate the episode.

We use the following *learning parameters* for all experiments: The tabular Q-learning agent was trained with a learning rate of $\alpha = 0.1$, a discount factor of $\gamma = 0.9$, and an initial exploration rate of $\epsilon = 0.4$ throughout all experiments. We chose an exponential epsilon decay of $\epsilon' = 0.9999 \cdot \epsilon$ with an update after every learning episode.

MDP Learning. To get the observations for MDP learning, we perform an abstraction over the states via the function Z. Given a concrete state (x, y), the function $Z((x, y))$ maps to a pair $(\mathit{terr}, \mathit{Pit})$, where *terr* is the terrain of the tile at (x, y) and *Pit* is a set of propositions denoting whether a pit is located in the neighboring tiles in each of the four cardinal directions.

For example, if the agent is at the coordinate $(1, 0)$ in Figure 2, with the origin of coordinates on the bottom left, the abstract observation would be $Z((1, 0)) = (c, \{\textit{pit-right}\})$. The terrain is c and there is a pit on the right.

Shield Construction. In all experiments, visiting a pit represents a safety violation. This property can be represented in LTL as follows: $\varphi = \mathbf{G}(\neg pit)$. Using this specification φ and a deterministic labeled MDP $\mathcal{M}_{\bigcup}$, we compute a shield $\pi_{\bigcup}$ using a relative threshold $\lambda_{sh} = 0.95$ and a finite horizon of $h = 2$. Thus, for any given state, the resulting shield $\pi_{\bigcup}$ allows an action a if the probability of not falling into a pit within the next 2 steps is at least $0.95 \cdot \alpha$, with α being the probability of not falling into a pit when taking the optimal action a'.

5.2 Experimental Results

In the following, we report on the performance of RL agents augmented with learned shields compared to the performance of unshielded RL agents.

To account for the stochastic nature of the environment and RL, we repeat every experiment 30 times. For each experiment, the number of iterations is set to $n_{iter} = 30$, the number of training episodes per iteration is $n_{episodes} = 1000$, and the maximal length of a training episode is set to $t_{max} = 200$ steps.

To evaluate the learned policies, we execute the policies at various stages throughout training and compare their performance. For every iteration, i.e., after every 1000^{th} training episode, we evaluate the intermediate policies of the agents by setting the exploration rate ϵ to 0 and performing 1000 episodes. Over these 1000 episodes, we compute the average cumulative reward of the intermediate policy. We refer to this value as *return*. For shielded agents, their corresponding shields are used during the intermediate executions for evaluation.

Figure 5, Figure 7, and Figure 9 show plots of the results of this evaluation, where the x-axes display the episodes and the y-axes display the return. The average performance of the *shielded agents* is represented by a thick *green line*, whereas a thick *red* line represents the average performance of *unshielded agents*. The light green and light red areas depict the range between the minimum and maximum performance of the shielded and the unshielded agents, respectively. Figure 6, Figure 8, and Figure 10 depict the number of safety violations throughout training, where the x-axes display the episodes and the y-axes display the number of times the agent visited a pit. The average number of safety violations of the *shielded agents* are represented by a thick *green line*, whereas a thick *red* line represents the average number of safety violations of *unshielded agents*. The light green and light red areas depict the range between the minimum and maximum number of safety violations of the shielded and the unshielded agents, respectively. In the following, we discuss the results from the experiments with the different gridworld shapes.

5.3 Zigzag Gridworlds

We start by discussing the performance of the RL agents in the *zigzag* gridworlds; see Figure 2 for the smallest such environment. Figure 5 shows the return, and Figure 6 shows the number of safety violations at various stages of RL.

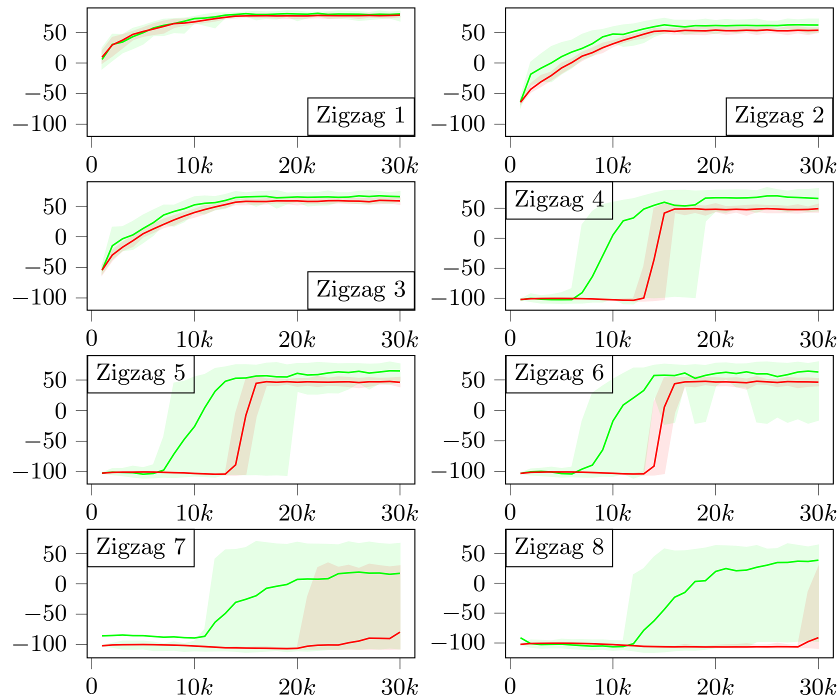

Figure 5. The return gained by intermediate policies throughout reinforcement learning in the *zigzag* gridworlds. The x-axes display the return and the y-axes display the episodes at which policies are evaluated. The green plots represent shielded performance, whereas the red plots represent unshielded performance.

The experiments in Figure 5 show almost identical returns and number of safety violations for shielded and unshielded agents for the three smallest gridworld instances. Starting with the fourth-smallest gridworld, shielded RL performs better on average. Initially, during the first episodes of both RL configurations, the average return is approximately -100, which is the penalty for falling into a pit. This means that the agent consistently falls into pits in the early stages of learning. After approximately 7 iterations, i.e, at episode 7000, the shielded agents start to reach the goal states, which leads to an increase in the return and a decrease in the number of safety violations. Unshielded agents need about twice the time to reach the goal location. Similar observations can be made for the next larger environments *Zigzag 5* and *Zigzag 6*, too. For the two largest *zigzag* gridworlds, unshielded RL fails to consistently reach the goal after $30,000$ training episodes. These two environments require relatively long paths to be traversed and the gained rewards are sparse. Hence, learned safety shields may benefit RL in environments with sparse rewards, where safety violations may prevent the agents from visiting states that give a positive reward.

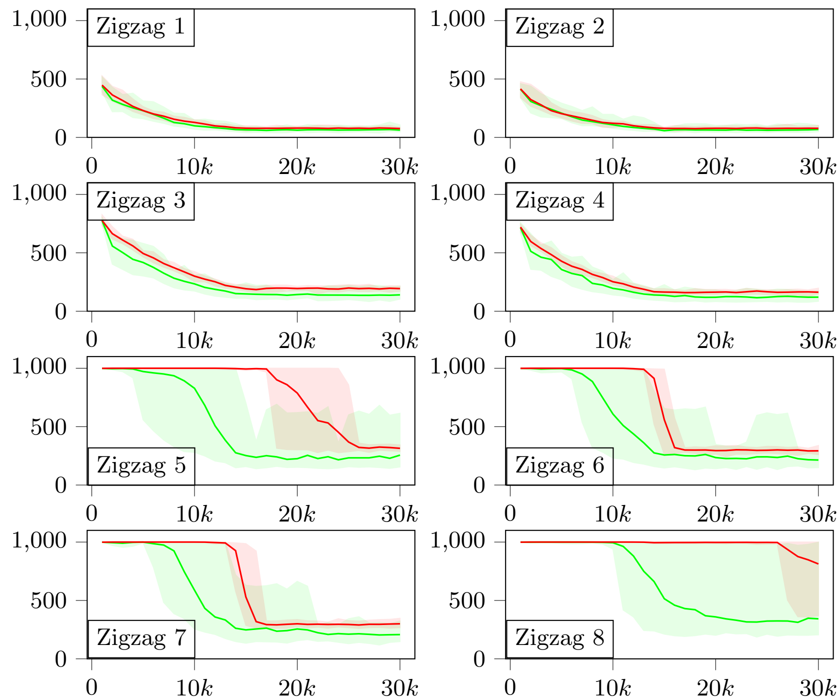

Figure 6. The number of safety violations throughout RL in the *zigzag* gridworlds. The x-axes display the number of violations and the y-axes display the episodes at which the policies are evaluated. The green plots represent the number of violations under the shielded policies, whereas the red plots represent the same under unshielded policies.

The decreases in the number of safety violations, as shown in Figure 6, match the observations on the performance increases illustrated in Figure 5.

On the negative side, note that the growth of the return is steeper for unshielded RL. For example, considering the environment *Zigzag 4*, it takes about 10000 episodes to reach an average return greater than 0 in the shielded case and it takes 15000 episodes in the unshielded case. Hence, there are 3000 episodes between first reaching the goal and reaching it more consistently for shielded RL, whereas unshielded RL only requires 1000 episodes to make this jump in performance.

Furthermore, consider the range between the minimum and the maximum return that is depicted by the shaded areas in the figures. The minimum return of unshielded RL is often lower than the minimum return of shielded RL even though it performs better on average. There is more variance in the return obtained by shielded RL. Also, the range between the minimum and the maximum number of safety violations is high for *Zigzag 8* until the end of train-

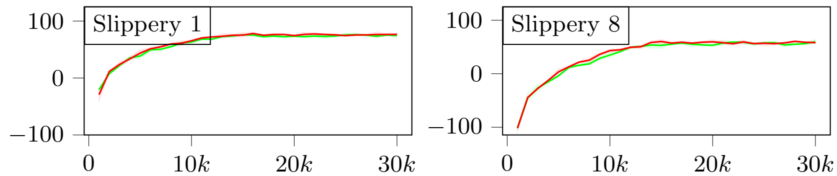

Figure 7. Return gained by intermediate policies throughout RL in the *slippery shortcuts* gridworlds. The x-axes display the return and the y-axes display the episodes at which policies are evaluated. Green plots: shielded agent, red plots: unshielded agent.

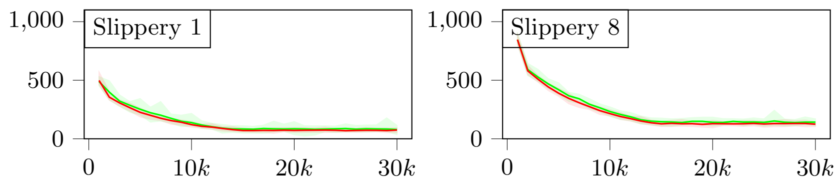

Figure 8. The number of safety violations throughout RL in the *slippery shortcuts* gridworlds. The x-axes display the number of violations and y-axes display the episodes at which the policies are evaluated. Green plots: shielded agent, red plots: unshielded agent.

ing. This could result from learning of MDPs that do not sufficiently capture safety-relevant information. We leave a closer investigation to future work.

5.4 Slippery Shortcuts Gridworlds

Next, we examine the performance of shielded and unshielded RL in the *slippery shortcuts* gridworlds illustrated in Figure 3. Figure 7 shows the average returns, and Figure 8 shows the number of safety violations gained by RL agents throughout learning. In contrast to the *zigzag* gridworlds, there is hardly any difference between the shielded and the unshielded configurations for any size of the environment, neither for the return nor for the number of safety violations. Therefore, we only printed the instances of *slippery 1* and *slippery 8* to safe space. Moreover, there is very little variability, as the minimum and maximum returns (safety violations) are very close to their average. Hence, performance is mostly governed by RL and shielding has little influence. In these environments, the agents succeed in finding safe paths without requiring assistance from a shield. This may result from the pits being farther away from optimal paths, as compared to the *zigzag* gridworlds.

5.5 Wall Gridworlds

In the following, we discuss the experiments performed on the *walls* gridworlds of Figure 4. Figure 9 shows the returns, and Figure 10 shows the number of

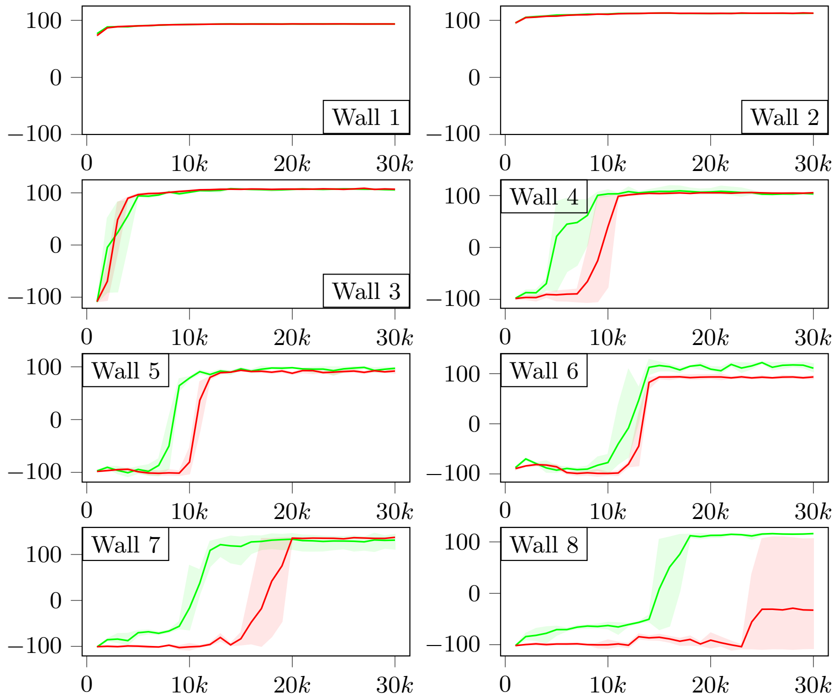

Figure 9. Return gained by intermediate policies throughout RL in the *walls* gridworlds. The x-axes display the return and the y-axes display the episodes at which policies are evaluated. Green plots: shielded agent, red plots: unshielded agent.

safety violations gained at different stages of learning. As for the *zigzag* gridworlds, we see hardly any difference between shielded and unshielded RL for the three smallest environments, whereas shielded RL performs better in the larger environments. Unlike before, however, there is less variability in the performance and number of safety violations of shielded RL. In Figure 4, we can see that the slippery tiles are farther away from the optimal route, which is also true for the larger *walls* gridworlds. As a result, learned MDPs do not need to be as accurate with respect to probability estimations for effective shields to be created. Especially for the largest example, *Wall 8*, shielding improves performance and reduces the number of safety violations considerably. It takes about 18000 episodes to learn a policy that consistently reaches the goal in all 30 repetitions of the corresponding experiments. In contrast, unshielded RL fails to consistently find a good policy even after 30000 episodes.

Finally, let us investigate a potential reason for performance improvements resulting from shielding or the absence thereof. In Figure 11, we show the size of learned MDPs compared to the size of the *walls* environments, i.e., the number of tiles in every environment. Since the environment size is constant in an experiment, it is shown as a black straight line. The learned MDP size, measured

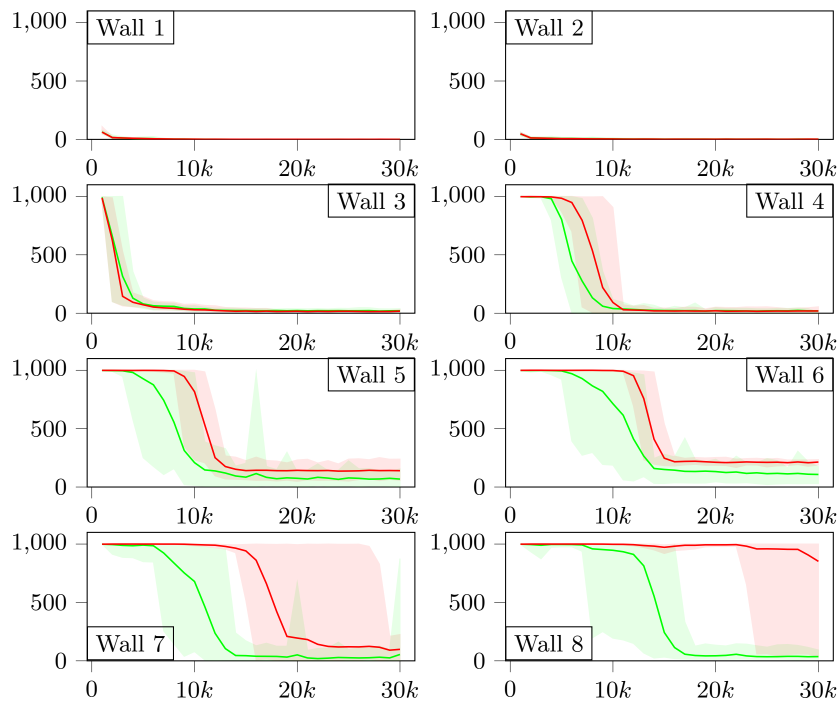

Figure 10. The number of safety violations throughout RL in the *walls* gridworlds. The x-axes display the number of violations and y-axes display the episodes at which the policies are evaluated. Green plots: shielded agent, red plots: unshielded agent.

in the number of states, generally increases throughout the learning due to more information getting available. It can be seen that for the first three environment sizes, the final learned MDPs are slightly larger than the environment. Hence, these MDPs cannot represent the environments and their safety-relevant features more efficiently than a Q-table. The fact that learned MDPs are even larger than the environments from which they are learned results from two properties of our learning setup. First, MDPs learned by IOAlergia only converge in the large sample limit to the true underlying MDPs [24]. There are no guarantees for MDPs learned from finite amounts of data. Second and more importantly, abstraction introduces non-determinism, while MDP learning basically performs a determinization of the resulting non-deterministic MDP. This determinization causes the number of states to increase, similar to the construction of belief MDPs from partially observable MDPs [9].

When the environment is larger than the learned MDP modeling safety-relevant features of the environment, shielding improves RL performance. This holds for all environments from *Wall 4* throughout *Wall 8*. Comparing Figure 9 or Figure 10 with Figure 11, we can see that the larger the size difference between

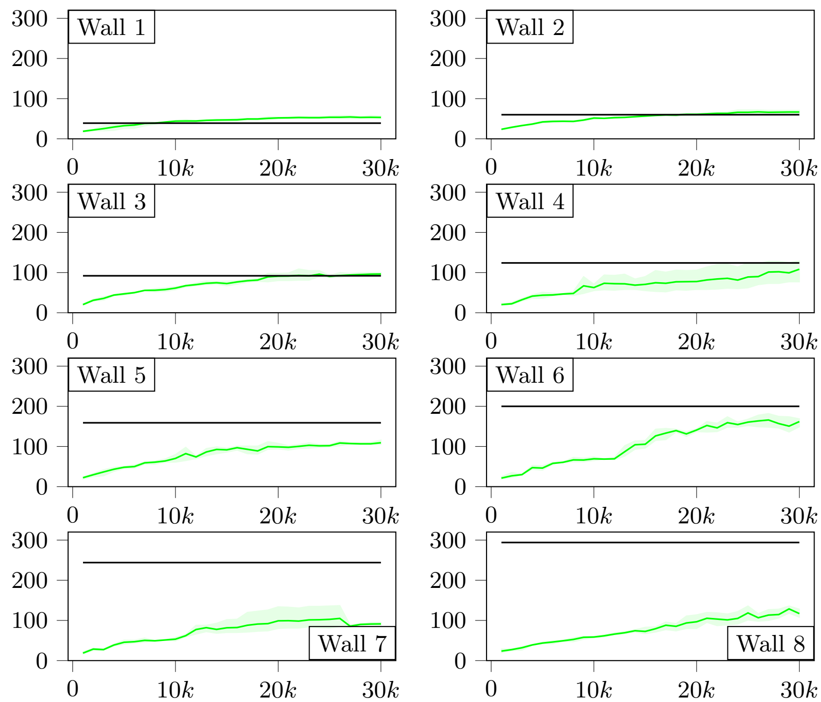

Figure 11. Average size of learned MDPs for the *walls* gridworlds (x-axes) plotted in green compared to the size of the environment plotted as a black line. The y-axes display the episode at which the MDP size was measured.

the environment and the learned MDP is, the larger the performance impact or reduction in the number of safety violations.

5.6 Discussion

We conclude this section with a discussion of the main results of our experiments and some insights that we gained. Our results show that learned shields can improve RL performance as illustrated by our first and third set of experiments. In the first case of the *zigzag* gridworlds, the agent has to traverse along tiles located closely to pits in order to reach the goal. Therefore, a shield is able to prevent many safety violations. In the case of the *walls* gridworlds, we observed that learning shields especially pays off when the learned safety MDP is much smaller than the complete environment. We will explore this connection in future work, as it may enable scaling to larger environments. Deep reinforcement learning can efficiently solve tasks in complex environments, such as, computer games [27], while we can abstract away non-safety-relevant details to learn small MDPs for shielding.

In the *slippery shortcut* gridworlds, we observed that shielding does not necessarily improve performance. It seems easier for the agent to infer through RL how to navigate safely than in the other examples.

6 Conclusion

We presented an approach for iterative safe reinforcement learning via learned shields. At runtime, we learn environmental models from collected traces and continuously update shields that prevent safety violations during execution. RL with learned shields comprises three steps: (1) An RL agent exploring the environment, protected by a shield, and collecting abstracted experiences, which represent safety-information about the environment. (2) Learning a deterministic labeled MDP from the collected data of the RL agent. (3) Synthesizing a shield from such an MDP.

In contrast to most previous work on shielding, which commonly requires abstract environment models, the proposed approach is model-free and therefore applicable in black-box environments. We learn environment models solely from experiences of the RL agent. The agent can also infer its policy using a model-free approach, such as, Q-learning [41]. The downside is that we cannot enforce absolute safety. In order to learn safety-relevant information, the agent needs to experience some safety violations. Despite this limitation, our evaluation shows that in most cases, RL with learned shields converges more quickly than unshielded RL. Since optimal policies in our experiments should inflict hardly any safety violations, faster convergence implies that shielded agents run into fewer safety violations.

In future work, we will explore RL with learned shields in environments of larger size, where we aim to combine deep RL with MDP learning. Our intuition is that we can generally represent safety-relevant environmental features concisely with an MDP over abstract observations. We expect this to be true even if, for instance, the agent perceives its environment by processing high-resolution images. In addition to other (deep) RL techniques, we will explore different automata learning techniques. For instance, we could integrate RL more directly into active automata learning of stochastic system models [35,37]. By learning timed automata [25,39,36,2], we could extend learning-based shielding to systems with time-dependent behaviour.

Acknowledgments. This work has been supported by the "University SAL Labs" initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems. Additionally, this project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N∘ 956123 - FOCETA.

References

1. Aichernig, B.K., Mostowski, W., Mousavi, M.R., Tappler, M., Taromirad, M.: Model learning and model-based testing. In: Bennaceur, A., Hähnle, R., Meinke, K. (eds.) Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers. Lecture Notes in Computer Science, vol. 11026, pp. 74–100. Springer (2018), https://doi.org/10.1007/978-3-319-96562-8_3
2. Aichernig, B.K., Pferscher, A., Tappler, M.: From passive to active: Learning timed automata efficiently. In: Lee, R., Jha, S., Mavridou, A. (eds.) NASA Formal Methods - 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11-15, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12229, pp. 1–19. Springer (2020), https://doi.org/10.1007/978-3-030-55754-6_1
3. Aichernig, B.K., Tappler, M.: Probabilistic black-box reachability checking (extended version). Formal Methods Syst. Des. **54**(3), 416–448 (2019)
4. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: Proceedings of the 32nd International Conference on Artificial Intelligence, AAAI 2018, New Orleans, Louisiana, USA, February 2-7, 2018. vol. 32, pp. 2669–2678. AAAI Press (2018), https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17211
5. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (2008)
6. Bloem, R., Könighofer, B., Könighofer, R., Wang, C.: Shield synthesis: - runtime enforcement for reactive systems. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9035, pp. 533–548. Springer (2015), https://doi.org/10.1007/978-3-662-46681-0_51
7. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI gym. CoRR **abs/1606.01540** (2016)
8. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Carrasco, R.C., Oncina, J. (eds.) Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994, Proceedings. Lecture Notes in Computer Science, vol. 862, pp. 139–152. Springer (1994), http://dx.doi.org/10.1007/3-540-58473-0_144
9. Cassandra, A.R., Kaelbling, L.P., Littman, M.L.: Acting optimally in partially observable stochastic domains. In: Hayes-Roth, B., Korf, R.E. (eds.) Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2. pp. 1023–1028. AAAI Press / The MIT Press (1994), http://www.aaai.org/Library/AAAI/1994/aaai94-157.php
10. Cobleigh, J.M., Giannakopoulou, D., Pasareanu, C.S.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2619, pp. 331–346. Springer (2003), https://doi.org/10.1007/3-540-36577-X_24
11. Corsi, D., Marchesini, E., Farinelli, A.: Formal verification of neural networks for safety-critical tasks in deep reinforcement learning. In: de Campos, C.P., Maathuis, M.H., Quaeghebeur, E. (eds.) UAI. Proceedings of Machine Learning Research,

vol. 161, pp. 333–343 (2021), https://proceedings.mlr.press/v161/corsi21a.html

12. Fu, J., Topcu, U.: Probably approximately correct MDP learning and control with temporal logic constraints. In: Fox, D., Kavraki, L.E., Kurniawati, H. (eds.) Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014 (2014), http://www.roboticsproceedings.org/rss10/p39.html
13. Furelos-Blanco, D., Law, M., Russo, A., Broda, K., Jonsson, A.: Induction of subgoal automata for reinforcement learning. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020. pp. 3890–3897. AAAI Press (2020), https://ojs.aaai.org/index.php/AAAI/article/view/5802
14. Gaon, M., Brafman, R.I.: Reinforcement learning with non-markovian rewards. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020. pp. 3980–3987. AAAI Press (2020), https://ojs.aaai.org/index.php/AAAI/article/view/5814
15. Giacobbe, M., Hasanbeig, M., Kroening, D., Wijk, H.: Shielding atari games with bounded prescience. In: Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2021, Virtual Event, United Kingdom, May 3-7, 2021. pp. 1507–1509. ACM (2021), https://dl.acm.org/doi/10.5555/3463952.3464141
16. Hasanbeig, M., Jeppu, N.Y., Abate, A., Melham, T., Kroening, D.: Deepsynth: Automata synthesis for automatic task segmentation in deep reinforcement learning. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. pp. 7647–7656. AAAI Press (2021), https://ojs.aaai.org/index.php/AAAI/article/view/16935
17. Howar, F., Steffen, B.: Active automata learning in practice - an annotated bibliography of the years 2011 to 2016. In: Bennaceur, A., Hähnle, R., Meinke, K. (eds.) Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers. Lecture Notes in Computer Science, vol. 11026, pp. 123–148. Springer (2018), https://doi.org/10.1007/978-3-319-96562-8_5
18. Icarte, R.T., Waldie, E., Klassen, T.Q., Valenzano, R.A., Castro, M.P., McIlraith, S.A.: Learning reward machines for partially observable reinforcement learning. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. pp. 15497–15508 (2019), https://proceedings.neurips.cc/paper/2019/hash/532435c44bec236b471a47a88d63513d-Abstract.html
19. Jansen, N., Könighofer, B., Junges, S., Serban, A., Bloem, R.: Safe reinforcement learning using probabilistic shields (invited paper). In: Konnov, I., Kovács, L. (eds.) 31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference). LIPIcs, vol. 171, pp. 3:1–3:16.

Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020), https://doi.org/10.4230/LIPIcs.CONCUR.2020.3
20. Kiran, B.R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A.A., Yogamani, S., Pérez, P.: Deep reinforcement learning for autonomous driving: A survey. IEEE Transactions on Intelligent Transportation Systems (2021)
21. Könighofer, B., Lorber, F., Jansen, N., Bloem, R.: Shield synthesis for reinforcement learning. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12476, pp. 290–306. Springer (2020), https://doi.org/10.1007/978-3-030-61362-4_16
22. Könighofer, B., Rudolf, J., Palmisano, A., Tappler, M., Bloem, R.: Online shielding for stochastic systems. In: Dutle, A., Moscato, M.M., Titolo, L., Muñoz, C.A., Perez, I. (eds.) NASA Formal Methods - 13th International Symposium, NFM 2021, Virtual Event, May 24-28, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12673, pp. 231–248. Springer (2021), https://doi.org/10.1007/978-3-030-76384-8_15
23. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning Markov decision processes for model checking. In: Fahrenberg, U., Legay, A., Thrane, C.R. (eds.) Proceedings Quantities in Formal Methods, QFM 2012, Paris, France, 28 August 2012. EPTCS, vol. 103, pp. 49–63 (2012), http://dx.doi.org/10.4204/EPTCS.103.6
24. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning deterministic probabilistic automata from a model checking perspective. Machine Learning **105**(2), 255–299 (2016), http://dx.doi.org/10.1007/s10994-016-5565-9
25. Mediouni, B.L., Nouri, A., Bozga, M., Bensalem, S.: Improved learning for stochastic timed models by state-merging algorithms. In: Barrett, C.W., Davies, M., Kahsai, T. (eds.) NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10227, pp. 178–193 (2017), https://doi.org/10.1007/978-3-319-57288-8_13
26. Meinke, K., Sindhu, M.A.: Incremental learning-based testing for reactive systems. In: Gogolla, M., Wolff, B. (eds.) Tests and Proofs - 5th International Conference, TAP@TOOLS 2011, Zurich, Switzerland, June 30 - July 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6706, pp. 134–151. Springer (2011), https://doi.org/10.1007/978-3-642-21768-5_11
27. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR **abs/1312.5602** (2013), http://arxiv.org/abs/1312.5602
28. Muskardin, E., Aichernig, B.K., Pill, I., Pferscher, A., Tappler, M.: Aalpy: An active automata learning library. In: Hou, Z., Ganesh, V. (eds.) Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12971, pp. 67–73. Springer (2021), https://doi.org/10.1007/978-3-030-88885-5_5
29. Muskardin, E., Tappler, M., Aichernig, B.K., Pill, I.: Reinforcement learning under partial observability guided by learned environment models. CoRR **abs/2206.11708** (2022), https://doi.org/10.48550/arXiv.2206.11708

30. Nouri, A., Raman, B., Bozga, M., Legay, A., Bensalem, S.: Faster statistical model checking by means of abstraction and learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8734, pp. 340–355. Springer (2014), https://doi.org/10.1007/978-3-319-11164-3_28
31. Peled, D.A., Vardi, M.Y., Yannakakis, M.: Black box checking. J. Autom. Lang. Comb. **7**(2), 225–246 (2002), https://doi.org/10.25596/jalc-2002-225
32. Pranger, S., Könighofer, B., Posch, L., Bloem, R.: TEMPEST - synthesis tool for reactive systems and shields in probabilistic environments. In: Hou, Z., Ganesh, V. (eds.) Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12971, pp. 222–228. Springer (2021), https://doi.org/10.1007/978-3-030-88885-5_15
33. Pranger, S., Könighofer, B., Tappler, M., Deixelberger, M., Jansen, N., Bloem, R.: Adaptive shielding under uncertainty. In: 2021 American Control Conference, ACC 2021, New Orleans, LA, USA, May 25-28, 2021. pp. 3467–3474. IEEE (2021), https://doi.org/10.23919/ACC50511.2021.9482889
34. Sutton, R.S., Barto, A.G.: Reinforcement learning - an introduction. Adaptive computation and machine learning, MIT Press (1998), https://www.worldcat.org/oclc/37293240
35. Tappler, M., Aichernig, B.K., Bacci, G., Eichlseder, M., Larsen, K.G.: l^*-based learning of markov decision processes (extended version). Formal Aspects Comput. **33**(4-5), 575–615 (2021), https://doi.org/10.1007/s00165-021-00536-5
36. Tappler, M., Aichernig, B.K., Larsen, K.G., Lorber, F.: Time to learn - learning timed automata from tests. In: André, É., Stoelinga, M. (eds.) Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27-29, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11750, pp. 216–235. Springer (2019), https://doi.org/10.1007/978-3-030-29662-9_13
37. Tappler, M., Muskardin, E., Aichernig, B.K., Pill, I.: Active model learning of stochastic reactive systems. In: Calinescu, R., Pasareanu, C.S. (eds.) Software Engineering and Formal Methods - 19th International Conference, SEFM 2021, Virtual Event, December 6-10, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13085, pp. 481–500. Springer (2021), https://doi.org/10.1007/978-3-030-92124-8_27
38. Vaandrager, F.W.: Model learning. Commun. ACM **60**(2), 86–95 (2017), https://doi.org/10.1145/2967606
39. Verwer, S., de Weerdt, M., Witteveen, C.: A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In: Sempere, J.M., García, P. (eds.) Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6339, pp. 203–216. Springer (2010), https://doi.org/10.1007/978-3-642-15488-1_17
40. Waga, M., Castellano, E., Pruekprasert, S., Klikovits, S., Takisaka, T., Hasuo, I.: Dynamic shielding for reinforcement learning in black-box environments. CoRR **abs/2207.13446** (2022). https://doi.org/10.48550/arXiv.2207.13446, https://doi.org/10.48550/arXiv.2207.13446
41. Watkins, C.J.C.H., Dayan, P.: Technical note q-learning. Mach. Learn. **8**, 279–292 (1992), https://doi.org/10.1007/BF00992698

42. Xu, Z., Gavran, I., Ahmad, Y., Majumdar, R., Neider, D., Topcu, U., Wu, B.: Joint inference of reward machines and policies for reinforcement learning. In: Beck, J.C., Buffet, O., Hoffmann, J., Karpas, E., Sohrabi, S. (eds.) Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020. pp. 590–598. AAAI Press (2020), https://ojs.aaai.org/index.php/ICAPS/article/view/6756